REMARKS

Examiner rejected claims 1-4 under 35 U.S.C. §101 as being directed to non-statutory subject matter. Examiner said that the claims preempt every substantial practical application of the idea that is embodied by the claims. Applicants do no such thing because ideas are not patentable, only devices, methods, compositions of matters, and improvements thereto. Applicants submit that claims 1-2 concern devices, which are patentable subject matter, and that the components of said devices are described with particularity. Applicants submit that claims 3-4 concern methods, which are patentable subject matter, and that the steps of said methods are described with particularity. Examiner also said that Applicants' device and method are so broad and sweeping as to cover both known and unknown uses of a pseudo-random bit sequence. Applicants do no such thing. Applicants submit that claims 1-4 only cover the particular devices and methods described in claims 1-4 and nothing else, that any use of a particular pseudo-random bit sequence is not covered by claims 1-4, and that only the particular devices and methods described in claims 1-4 are covered. Examiner suggested amending the claims to include specific implementation of which the claimed invention may be used. Applicants submit that the claims are already specific as to what components make up the devices, what steps make up the methods, and what the devices and methods are used for (i.e., generating an uncorrelated pseudo-random bit sequence uniformly distributed over a user-definable value K, where K+1 has *m* prime factors).

Per Examiner's request, Applicants add claims 5 and 6 to include an application of the invention for cryptographic systems.

Examiner rejected claims 3-4 under 35 U.S.C. §103(a) as being unpatentable over U.S. Pat. No. 5,446,683 (Mullen) in view of U.S. Pat. No. 5,974,144 (Brandman).

With regard to claim 3, Examiner said that Mullen discloses a method of generating an uncorrelated pseudo-random bit sequence by generating m pseudo-random bit sequences, r1, r2,…,rm, and generating the uncorrelated pseudo-random sequence. Mullen does not generate an uncorrelated pseudo-random sequence but instead generates a series of pseudo-random sequences using a series of Linear Feedback Shift Registers (LFSRs). As evidenced by the attached Wikipedia definition of an LFSR, an LFSR consists of a number of flip-flop registers and exclusive-or, or XOR, gates. As evidenced by the attached Wikipedia definition of XOR, an XOR performs addition modulo two. Therefore, it does not perform multiplication of prime factors and pseudo-random sequences (Claim 3, page 13, line 4) as do Applicants.

Examiner admits that Mullen does not disclose selecting a user-definable value K, where K is a positive integer; factoring K+1 into *m* prime factors $q_1, q_2,\ldots,q_m$; using pseudo-random sequences uniformly distributed over a range $(0,\ldots q_{i-1})$, where i = 1,2,…,*m*; or generating an uncorrelated pseudo-random sequence $R = r_1+q_1r_2+q_1q_2r_3+\ldots+q_1q_2\ldots q_{m-1}r_m$, as do Applicants.

However, Examiner said that Brandman discloses selecting a user-definable value K, where K is a positive integer; factoring K+1 into *m* prime factors $q_1, q_2,\ldots,q_m$; using pseudo-random sequences uniformly distributed over a range $(0,\ldots q_{i-1})$, where i = 1,2,…,*m*; and generating an uncorrelated pseudo-random sequence $R = r_1+q_1r_2+q_1q_2r_3+\ldots+q_1q_2\ldots q_{m-1}r_m$, as do Applicants.

Examiner's citation discloses no such thing. Instead Examiner's citation discloses the RSA public-key cryptographic method, which is nothing like Applicants' method of generating an uncorrelated pseudo-random bit sequence. As evidenced by the attached pages of *Applied Cryptography, 2nd Ed.*, the RSA algorithm discloses the steps of choosing two large prime numbers (i.e., *p*, *q*), multiplying them together (i.e., $n = pq$), choosing an encryption key *e* so that *e* is relatively prime to $(p-1)(q-1)$, and computing a decryption key *d* so that *ed* is equal to $(1 \pmod{(p-1)(q-1)})$. Then, a message *m* can be encrypted to produce ciphertext *c* by computing (m^e

mod *n)*. The plaintext *m* can then be recovered by computing (c^d mod *n*). The steps of the RSA algrithm are nothing like those of Applicants' Claim 3 (Claim 3, page 12 line 18- page 13, line 4).

Examiner then said that Brandman does not disclose using prime factors to generate an uncorrelated pseudo-random sequence $R = r_1+q_1r_2+q_1q_2r_3+\ldots+q_1q_2\ldots q_{m-1}r_m$, as do Applicants. Previously, Examiner said that it did. Therefore, Applicants request Examiner to state what he think Brandman discloses, because Brandman cannot both disclose generating an uncorrelated pseudo-random sequence $R = r_1+q_1r_2+q_1q_2r_3+\ldots+q_1q_2\ldots q_{m-1}r_m$ and not disclosing it.

Then Examiner said that it would have been obvious to one of ordinary skill in the art to incorporate prime factors into the sequence to achieve the same predicted result. First, Brandman does not disclose generating an uncorrelated pseudo-random sequence $R = r_1+q_1r_2+q_1q_2r_3+\ldots+q_1q_2\ldots q_{m-1}r_m$ as do Applicants. Second, Brandman discloses the RSA public-key algorithm, which is not a pseudo-random number generator. Third, adding prime factors to Brandman produces a modified RSA public-key algorithm, not Applicants' method. Fourth, it would not be obvious to anyone skilled in the art to modify the RSA public-key algorithm to arrive at Applicants' method.

Examiner then said that it would have been obvious to one of ordinary skill in the art to modify Mullen by incorporating prime factors and prime factorization in a cryptographic system, as taught by Brandman, to achieve a level of security in a pseudo-random sequence environment such as one used in the areas of encryption and cryptography. Mullen does not include multiplication as does Applicants' method. Therefore, Mullen could not use prime factors to produce a pseudo-random number as do Applicants. In addition, Applicants do not involve achieving any level of security for cryptography, but is only concerned with producing an uncorrelated pseudo-random number. Therefore, is would not have been obvious to incorporate prime factors into Mullen, because that would destroy the intent of Mullen.

With regard to claim 4, Examiner said that Mullen and Brandman does not disclose that $q_1, q_2,\ldots,q_m$ are ordered from smallest value q_1 to largest value q_m. However, Examiner took

official notice that it would have been obvious to one skilled in the art to manipulate the factors in this way to achieve the predicted result of an uncorrelated pseudo-random sequence. Applicants challenges Examiner's factual assertion as not properly officially noticed and not properly based upon common knowledge. Examiner's error includes the fact that Mullen does not include multiplication but only addition as described above, that Brandman discloses the RSA public-key cryptographic algorithm and not a pseudo-random number generator, that providing prime factors to Mullen would not result in the pseudo-random number generated by Applicants.

Applicants hereby amend their application per Examiner's request for a set of claims concerning a cryptographic application. However, since Applicants' invention is not limited to cryptographic applications, Applicants retain claims 1 and 2.

A new fee determination is provided that shows that Applicants need not provide any additional fee to have these added claims examined.

Reconsideration of the application in light of the amendment and the remarks is requested. Applicants request Examiner withdraw his rejections and allow claims 1-6.

Respectfully submitted,

Robert D. Morelli
Registration No. 37,398
(301) 688-0287

Linear feedback shift register

From Wikipedia, the free encyclopedia

A **linear feedback shift register** (LFSR) is a shift register whose input bit is a linear function of its previous state.

The only linear functions of single bits are xor and inverse-xor; thus it is a shift register whose input bit is driven by the exclusive-or (xor) of some bits of the overall shift register value.

The initial value of the LFSR is called the seed, and because the operation of the register is deterministic, the sequence of values produced by the register is completely determined by its current (or previous) state. Likewise, because the register has a finite number of possible states, it must eventually enter a repeating cycle. However, a LFSR with a well-chosen feedback function can produce a sequence of bits which appears random and which has a very long cycle.

Applications of LFSRs include generating pseudo-random numbers, pseudo-noise sequences, fast digital counters, and whitening sequences. Both hardware and software implementations of LFSRs are common.

Contents

Fibonacci LFSRs

The list of the bits positions that affect the next state is called the tap sequence. In the diagram below, the sequence is [16,14,13,11,0]. In a Fibonacci LFSR, as below, the taps are XOR'd sequentially with the output and then feed back into the leftmost bit.

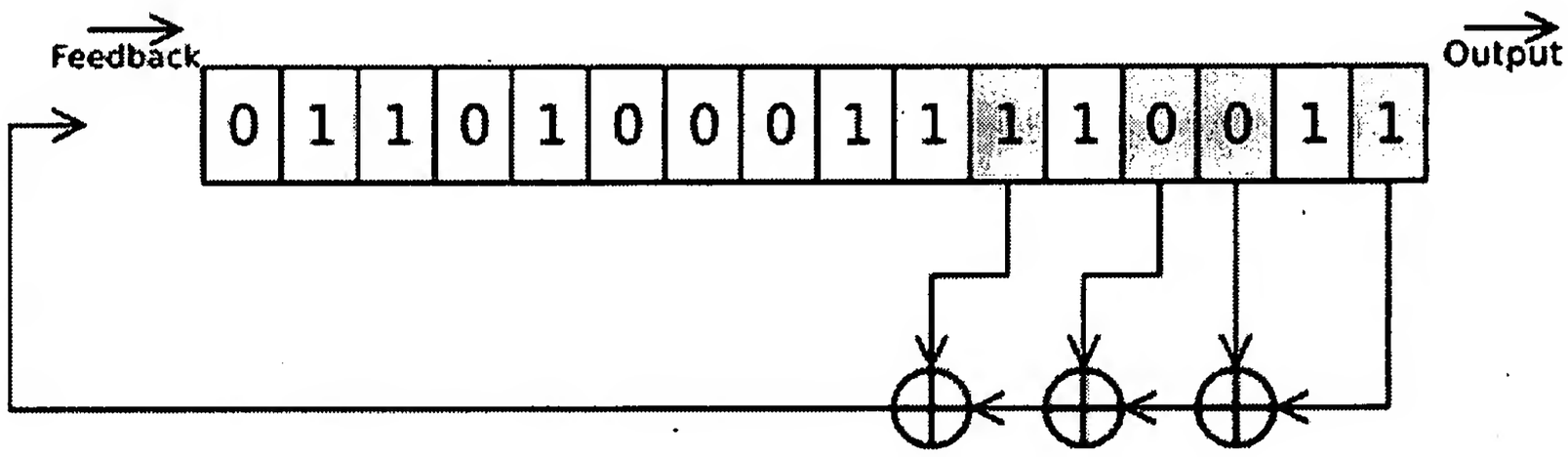
- The outputs that influence the input are called *taps* (blue in the diagram below).
- A maximal LFSR produces an n-sequence (i.e. cycles through all possible $2^n - 1$ states within the shift register except the state where all bits are zero), unless it contains all zeros, in which case it will never change.

The sequence of numbers generated by a LFSR can be considered a binary numeral system just as valid as Gray code or the natural binary code.

The tap sequence of an LFSR can be represented as a polynomial mod 2. This means that the coefficients of the polynomial must be 1's or 0's. This is called the feedback polynomial or characteristic polynomial. For example, if the taps are at the 16th, 14th, 13th and 11th bits (as below), the resulting LFSR polynomial is:

The 'one' in the polynomial does not correspond to a tap - it corresponds to the input to the first bit (i.e. x^0, which is equivalent to 1). The powers of the terms represent the tapped bits, counting from the left. The first and last bits are always connected as an input and tap respectively.

- If (and only if) this polynomial is a primitive, then the LFSR is maximal
- The LFSR will only be maximal if the number of taps is even
- There can be more than one maximal tap sequence for a given LFSR length
- Once one maximal tap sequence has been found, another automatically follows. If the tap sequence, in an n-bit LFSR, is [n,A,B,C,0], where the 0 corresponds to the $x^0 = 1$ term, then the corresponding 'mirror' sequence is [n,n-C,n-B,n-A,0]. So the tap sequence [32,3,2,0] has as its counterpart [32,30,29,0]. Both give a maximal sequence.



Output-stream properties

- Ones and zeroes occur in 'runs'. The output stream 0110100, for example consists of five runs of lengths 1,2,1,1,2, in order. In one period of a maximal LFSR, 2^{n-1} runs occur

(for example, a six bit LFSR will have 32 runs). Exactly 1 / 2 of these runs will be one bit long, 1 / 4 will be two bits long, up to a single run of zeroes $n - 1$ bits long, and a single run of ones n bits long. This same property is statistically expected in a truly random sequence.

- LFSR outputs streams are deterministic. If you know the present state, you can predict the next state. This is not possible with truly random events such as nuclear decay.
- The output stream is reversible; an LFSR with mirrored tap sequence will cycle through the states in reverse order.

A drop in replacement for Gray Code counters

Some applications need to mark individual locations along a certain distance with unique values. For example, most tape measures mark each inch or centimeter with a unique number using the decimal numeral system. When computer index or framing locations need to be machine-readable, they are often marked using a LFSR sequence, because LFSR counters are simpler and faster than any other kind of binary counter. LFSRs are faster than natural binary counters and Gray code counters. Given an output sequence you can construct a LFSR of minimal size by using the Berlekamp-Massey algorithm.

Galois LFSRs

Named after the French mathematician Évariste Galois, a *Galois* LFSR, or an LFSR in Galois configuration, is an alternate structure that can generate the same output sequences as a conventional LFSR. In the Galois configuration, when the system is clocked, bits that are not taps are shifted as normal to the next flip-flop. The taps, on the other hand, are XOR'd with the new output, which also becomes the new input. These won't be shifted in until the next clock cycle.

To generate the same output sequence, the order of the taps is the *counterpart* (see above) of the order for the conventional LFSR, otherwise the sequence will be in reverse. Note that the internal state of the LFSR is not necessarily the same. The Galois register above has the same output as the Fibonnacci register in the first section.

- Galois LFSRs do not concatenate every tap to produce the new input (the XOR'ing is done within the LFSR and no XOR gates are run in serial, therefore the propagation times are reduced to that of one XOR rather than a whole chain), thus it is possible for each tap to be computed in parallel, increasing the speed of execution.
- In a software implementation of an LFSR, the Galois form is more efficient as the XOR operations can be implemented a word at a time: only the output bit must be examined individually.

Below is example of 32-bit maximal period Galois LFSR simulated in C:

```
unsigned int lfsr = 1;
while(1)
  lfsr = (lfsr >> 1) ^ (-(signed int)(lfsr & 1) & 0xd0000001u); /* taps 32 31 29 1 */
```

Applications

LFSRs can be implemented in hardware, and this makes them useful in applications that require very fast generation of a pseudo-random sequence, such as direct-sequence spread spectrum radio.

The Global Positioning System uses a LFSR to rapidly transmit a sequence that indicates high-precision relative time offsets. The Nintendo Entertainment System video game console also has a LFSR as part of its sound system. ([1] (http://nocash.emubase.de/everynes.htm))

Uses in cryptography

LFSRs have long been used as a pseudo-random number generator for use in stream ciphers (especially in military cryptography), due to the ease of construction from simple electromechanical or electronic circuits, long periods, and very uniformly distributed outputs. However the outputs of LFSRs are completely linear, leading to fairly easy cryptanalysis.

Three general methods are employed to reduce this problem in LFSR based stream ciphers

- Non-linear combination of several bits from the LFSR state;
- Non-linear combination of the outputs of two or more LFSRs; or
- Irregular clocking of the LFSR, as in the alternating step generator.

Important LFSR-based stream ciphers include A5/1, A5/2, E0, and the shrinking generator.

Uses in digital broadcasting and communications

To prevent short repeating sequences (e.g., runs of 0's or 1's) from forming spectral lines that may complicate symbol tracking at the receiver or interfere with other transmissions, linear feedback registers are often used to "randomize" the transmitted bitstream. This randomization is removed at the receiver after demodulation. When the LFSR runs at the same rate as the transmitted symbol stream, this technique is referred to as scrambling. When the LFSR runs considerably faster than the symbol stream, expanding the bandwidth of the transmitted signal, this is direct-sequence spread spectrum.

Neither scheme should be confused with encryption or encipherment; scrambling and spreading with LFSRs do *not* protect the information from eavesdropping.

Digital broadcasting systems that use linear feedback registers

- ATSC Standards (HDTV transmission system – North America)
- DAB (Digital audio broadcasting system -- for radio)
- DVB-T (HDTV transmission system – Europe, Australasia)
- NICAM (digital audio system for television)

Other digital communications systems using LFSR:

- IBS (INTELSAT business service)
- IDR (Intermedaite Data Rate service)
- SDI (Serial Digital Interface transmission)
- Data transfer over PSTN (according to the ITU-T V-series recommendations)

See also

- Pinwheel
- Mersenne twister
- PN Sequences
- Maximum length sequence

External links

- International Telecommunications Union Recommendation O.151 (http://www.itu.int/rec/T-REC-O.151-199210-I/en) (August 1992)
- Maximal Length LFSR table (http://www.xilinx.com/bvdocs/appnotes/xapp052.pdf) with length from 3 to 168
- Maximal Length LFSR table (http://www.physics.otago.ac.nz/px/research/electronics/papers/technical-reports/lfsr_table.pdf) with length from 1 to 786, also 1024 and 2048.
- Pseudo-Random Number Generation Routine (http://www.maxim-ic.com/appnotes.cfm?appnote_number=1743&CMP=WP-9)
- http://www.ee.ualberta.ca/~elliott/ee552/studentAppNotes/1999f/Drivers_Ed/lfsr.html
- http://www.quadibloc.com/crypto/co040801.htm
- Simple explanation of LFSRs for Engineers (http://www.yikes.com/~ptolemy/lfsr_web/index.htm)
- Feedback terms (http://www.ece.cmu.edu/~koopman/lfsr/index.html)
- General LFSR Theory (http://homepage.mac.com/afj/lfsr.html)
- Table of Maximal Tap Sequences (http://homepage.mac.com/afj/taplist.html)
- Shift register code generator (http://www-rocq.inria.fr/codes/LFSR/index.html)

Retrieved from "http://en.wikipedia.org/wiki/Linear_feedback_shift_register"

Categories: Digital registers | Cryptographic algorithms | Pseudorandom number generators

XOR gate

From Wikipedia, the free encyclopedia

The **XOR gate** is a digital logic gate that implements exclusive disjunction - it behaves according to the truth table to the right. A HIGH output (1) results if one, and only one, of the inputs to the gate is HIGH (1). If both inputs are LOW (0) or both are HIGH (1), a LOW output (0) results.

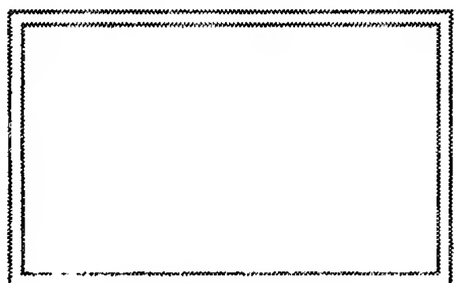
INPUT A	INPUT B	OUTPUT A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

This function is addition modulo 2. As a result, XOR gates are used to implement binary addition in computers. A half adder consists of an XOR gate and an AND gate.

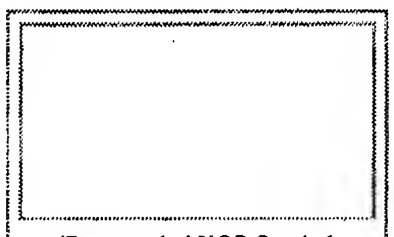
Contents

Symbols

There are two symbols for XOR gates: the 'military' symbol and the 'rectangular' symbol. For more information see Logic Gate Symbols



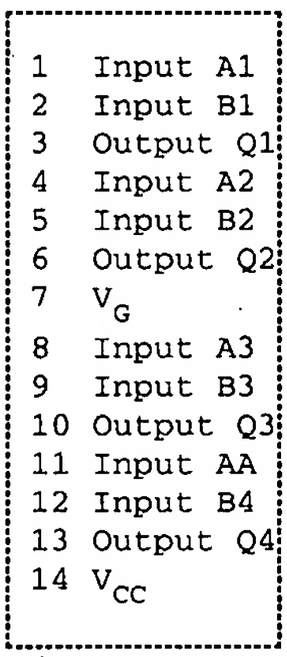
'Military' XOR Symbol



'Rectangular' XOR Symbol

Hardware description and pinout

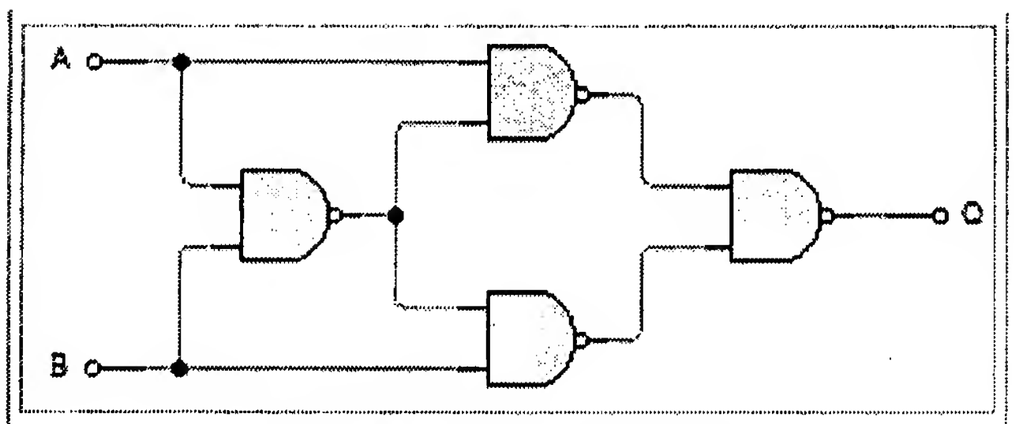
XOR gates are basic logic gates, and as such they are recognised in TTL and CMOS ICs. The standard, 4000 series, CMOS IC is the 4070, which includes four independent, two-input, XOR gates. The 4070 replaces the less reliable 4030, but keeps the pinout. The pinout diagram is as follows:



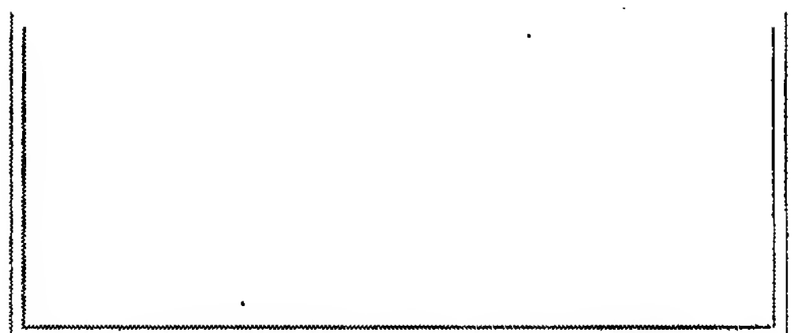
This device is available from most semiconductor manufacturers such as Philips. It is usually available in both through-hole DIL and SOIC format. Datasheets are readily available in most Datasheet Databases.

Alternatives

If no specific XOR gates are available, one can be made from four NAND or five NOR gates in the configurations shown below. Interestingly, any logic gate can be made from a combination of NAND gates or a combination of NOR gates.



XOR gate constructed using only NAND gates



XOR gate constructed using only NOR gates

More than two inputs

The XOR operation is a binary operation and is therefore defined only for two inputs. [1] It is nevertheless common in electronic design to talk of "XORing" three or more signals.

The most common interpretation of this usage is that the first two signals are fed into an XOR gate, then the output of that gate is fed into a second XOR gate together with the third signal, and so on for any remaining signals. The result is a circuit that outputs a 1 when the number of 1s at its inputs is odd, and a 0 when the number of incoming 1s is even. This makes it practically useful as a parity generator or a modulo-2 adder.

A second interpretation is also possible, based on both the linguistic sense of the term "exclusive OR" and the IEC symbol for an XOR gate (see right). This interpretation states that the output is 1 when one *or* other of the inputs, *exclusively*, is 1. The "=1" in the IEC symbol implies the same thing. However, the IEC symbol was not intended to be modified by adding further inputs, and becomes invalid when this is done. This interpretation is rarely used in electronics, since parity generators and adders are in more common use than "1 of n" detectors.

See also

- XNOR gate
- Boolean algebra (logic)
- Logic gates

References

1. ^ "exclusive OR n." *The Concise Oxford English Dictionary*, Eleventh edition revised . Ed. Catherine Soanes and Angus Stevenson. Oxford University Press, 2006. *Oxford Reference Online*. Oxford University Press. Birmingham City Council. 27 May 2007 [1] (http://www.oxfordreference.com/views/ENTRY.html?subview=Main&entry=t23.e19236)

APPL[illegible]PHY,
SECON[illegible]DITION

PROTOCOLS, ALGO[illegible]OURCE CODE IN C

John Wiley & Sons, Inc.
New York • Chichester • Brisbane • Toronto • Singapore

Other algorithms have been proposed that use ideas similar to those used in knapsack cryptosystems, but these too have been broken. The Lu-Lee cryptosystem [990,13] was broken in [20,614,873]; a modification [507] is also insecure [1620]. Attacks on the Goodman-McAuley cryptosystem are in [646,647,267,268]. The Pieprzyk cryptosystem [1246] can be broken by similar attacks. The Niemi cryptosystem [1169], based on modular knapsacks, was broken in [345,788]. A newer multistage knapsack [747] has not yet been broken, but I am not optimistic. Another variant is [294].

While a variation of the knapsack algorithm is currently secure—the Chor-Rivest knapsack [356], despite a "specialized attack" [743]—the amount of computation required makes it far less useful than the other algorithms discussed here. A variant, called the Powerline System, is not secure [958]. Most important, considering the ease with which all the other variations fell, it doesn't seem prudent to trust them.

Patents

The original Merkle-Hellman algorithm is patented in the United States [720] and worldwide (see Table 19.1). Public Key Partners (PKP) licenses the patent, along with other public-key cryptography patents (see Section 25.5). The U.S. patent will expire on August 19, 1997.

19.3 RSA

Soon after Merkle's knapsack algorithm came the first full-fledged public-key algorithm, one that works for encryption and digital signatures: RSA [1328,1329]. Of all the public-key algorithms proposed over the years, RSA is by far the easiest to understand and implement. (Martin Gardner published an early description of the algorithm in his "Mathematical Games" column in *Scientific American* [599].) It is

Table 19.1
Foreign Merkle-Hellman Knapsack Patents

Country	Number	Date of Issue
Belgium	871039	5 Apr 1979
Netherlands	7810063	10 Apr 1979
Great Britain	2006580	2 May 1979
Germany	2843583	10 May 1979
Sweden	7810478	14 May 1979
France	2405532	8 Jun 1979
Germany	2843583	3 Jun 1982
Germany	2857905	15 Jul 1982
Canada	1128159	20 Jul 1982
Great Britain	2006580	18 Aug 1982
Switzerland	63416114	14 Jan 1983
Italy	1099780	28 Sep 1985

also the most popular. Named after the three inventors—Ron Rivest, Adi Shamir, and Leonard Adleman—it has since withstood years of extensive cryptanalysis. Although the cryptanalysis neither proved nor disproved RSA's security, it does suggest a confidence level in the algorithm.

RSA gets its security from the difficulty of factoring large numbers. The public and private keys are functions of a pair of large (100 to 200 digits or even larger) prime numbers. Recovering the plaintext from the public key and the ciphertext is conjectured to be equivalent to factoring the product of the two primes.

To generate the two keys, choose two random large prime numbers, p and q. For maximum security, choose p and q of equal length. Compute the product:

$$n = pq$$

Then randomly choose the encryption key, e, such that e and $(p-1)(q-1)$ are relatively prime. Finally, use the extended Euclidean algorithm to compute the decryption key, d, such that

$$ed \equiv 1 \pmod{(p-1)(q-1)}$$

In other words,

$$d = e^{-1} \bmod ((p-1)(q-1))$$

Note that d and n are also relatively prime. The numbers e and n are the public key; the number d is the private key. The two primes, p and q, are no longer needed. They should be discarded, but never revealed.

To encrypt a message m, first divide it into numerical blocks smaller than n (with binary data, choose the largest power of 2 less than n). That is, if both p and q are 100-digit primes, then n will have just under 200 digits and each message block, m_i, should be just under 200 digits long. (If you need to encrypt a fixed number of blocks, you can pad them with a few zeros on the left to ensure that they will always be less than n. The encrypted message, c, will be made up of similarly sized message blocks, c_i, of about the same length. The encryption formula is simply

$$c_i = m_i^e \bmod n$$

To decrypt a message, take each encrypted block c_i and compute

$$m_i = c_i^d \bmod n$$

Since

$$c_i^d = (m_i^e)^d = m_i^{ed} = m_i^{k(p-1)(q-1)+1} = m_i m_i^{k(p-1)(q-1)} = m_i * 1 = m_i \text{ all } (\bmod n)$$

the formula recovers the message. This is summarized in Table 19.2.

The message could just as easily have been encrypted with d and decrypted with e; the choice is arbitrary. I will spare you the number theory that proves why this works; most current texts on cryptography cover it in detail.

A short example will probably go a long way to making this clearer. If $p = 47$ and $q = 71$, then

Table 19.2
RSA Encryption

Public Key:
n product of two primes, p and q (p and q must remain secret)
e relatively prime to $(p-1)(q-1)$

Private Key:
d $e^{-1} \bmod ((p-1)(q-1))$

Encrypting:
$c = m^e \bmod n$

Decrypting:
$m = c^d \bmod n$

$$n = pq = 3337$$

The encryption key, e, must have no factors in common with

$$(p-1)(q-1) = 46 * 70 = 3220$$

Choose e (at random) to be 79. In that case

$$d = 79^{-1} \bmod 3220 = 1019$$

This number was calculated using the extended Euclidean algorithm (see Section 11.3). Publish e and n, and keep d secret. Discard p and q.

To encrypt the message

$$m = 68823268796666 83$$

first break it into small blocks. Three-digit blocks work nicely in this case. The message is split into six blocks, m_i, in which

$$m_1 = 688$$
$$m_2 = 232$$
$$m_3 = 687$$
$$m_4 = 966$$
$$m_5 = 668$$
$$m_6 = 003$$

The first block is encrypted as

$$688^{79} \bmod 3337 = 1570 = c_1$$

Performing the same operation on the subsequent blocks generates an encrypted message:

$$c = 1570\ 2756\ 2091\ 2276\ 2423\ 158$$

Decrypting the message requires performing the same exponentiation using the decryption key of 1019, so

$$1570^{1019} \bmod 3337 = 688 = m_1$$

The rest of the message can be recovered in this manner.

RSA in Hardware

Much has been written on the subject of hardware implementations of RSA [1314, 1474,1456,1316,1485,874,1222,87,1410,1409,1343,998,367,1429,523,772]. Good survey articles are [258,872]. Many different chips perform RSA encryption [1310,252, 1101,1317,874,69,737,594,1275,1563,509,1223]. A partial list of currently available RSA chips, from [150,258], is listed in Table 19.3. Not all are available on the open market.

Speed of RSA

In hardware, RSA is about 1000 times slower than DES. The fastest VLSI hardware implementation for RSA with a 512-bit modulus has a throughput of 64 kilobits per second [258]. There are also chips that perform 1024-bit RSA encryption. Currently chips are being planned that will approach 1 megabit per second using a 512-bit modulus; they will probably be available in 1995. Manufacturers have also implemented RSA in smart cards; these implementations are slower.

In software, DES is about 100 times faster than RSA. These numbers may change slightly as technology changes, but RSA will never approach the speed of symmetric algorithms. Table 19.4 gives sample software speeds of RSA [918].

Software Speedups

RSA encryption goes much faster if you're smart about choosing a value of e. The three most common choices are 3, 17, and 65537 ($2^{16} + 1$). (The binary representation of 65537 has only two ones, so it takes only 17 multiplications to exponentiate.) X.509 recommends 65537 [304], PEM recommends 3 [76], and PKCS #1 (see Section 24.14) recommends 3 or 65537 [1345]. There are no security problems with using

Table 19.3
Existing RSA Chips

Company	Clock Speed	Baud Rate Per 512 Bits	Clock Cycles Per 512 Bit Encryption	Technology	Bits per Chip	Number of Transistors
Alpha Techn.	25 MHz	13 K	.98 M	2 micron	1024	180,000
AT&T	15 MHz	19 K	.4 M	1.5 micron	298	100,000
British Telecom	10 MHz	5.1 K	1 M	2.5 micron	256	——
Business Sim. Ltd.	5 MHz	3.8 K	.67 M	Gate Array	32	——
Calmos Syst. Inc.	20 MHz	28 K	.36 M	2 micron	593	95,000
CNET	25 MHz	5.3 K	2.3 M	1 micron	1024	100,000
Cryptech	14 MHz	17 K	.4 M	Gate Array	120	33,000
Cylink	30 MHz	6.8 K	1.2 M	1.5 micron	1024	150,000
GEC Marconi	25 MHz	10.2 K	.67 M	1.4 micron	512	160,000
Pijnenburg	25 MHz	50 K	.256 M	1 micron	1024	400,000
Sandia	8 MHz	10 K	.4 M	2 micron	272	86,000
Siemens	5 MHz	8.5 K	.03 M	1 micron	512	60,000

PTO/SB/06 (12-04)
Approved for use through 7/31/2006. OMB 0651-0032
U.S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE

PATENT APPLICATION FEE DETERMINATION RECORD

Substitute for Form PTO-875

Application or Docket Number: KUEHNEL3-1

APPLICATION AS FILED – PART I

FOR	(Column 1) NUMBER FILED	(Column 2) NUMBER EXTRA	SMALL ENTITY RATE ($)	SMALL ENTITY FEE ($)	OR	OTHER THAN SMALL ENTITY RATE ($)	OTHER THAN SMALL ENTITY FEE ($)
BASIC FEE (37 CFR 1.16(a), (b), or (c))	N/A	N/A	N/A			N/A	
SEARCH FEE (37 CFR 1.16(k), (i), or (m))	N/A	N/A	N/A			N/A	
EXAMINATION FEE (37 CFR 1.16(o), (p), or (q))	N/A	N/A	N/A			N/A	
TOTAL CLAIMS (37 CFR 1.16(i))	minus 20 =	*	x =		OR	x =	
INDEPENDENT CLAIMS (37 CFR 1.16(h))	minus 3 =	*	x =			x =	
APPLICATION SIZE FEE (37 CFR 1.16(s))	If the specification and drawings exceed 100 sheets of paper, the application size fee due is $250 ($125 for small entity) for each additional 50 sheets or fraction thereof. See 35 U.S.C. 41(a)(1)(G) and 37 CFR 1.16(s).						
MULTIPLE DEPENDENT CLAIM PRESENT (37 CFR 1.16(j))			N/A			N/A	
			TOTAL			TOTAL	

* If the difference in column 1 is less than zero, enter "0" in column 2.

APPLICATION AS AMENDED – PART II

AMENDMENT A

	(Column 1) CLAIMS REMAINING AFTER AMENDMENT		(Column 2) HIGHEST NUMBER PREVIOUSLY PAID FOR	(Column 3) PRESENT EXTRA	SMALL ENTITY RATE ($)	SMALL ENTITY ADDITIONAL FEE ($)	OR	OTHER THAN SMALL ENTITY RATE ($)	OTHER THAN SMALL ENTITY ADDITIONAL FEE ($)
Total (37 CFR 1.16(i))	* 6	Minus	** 20	= 0	x =		OR	x =	0
Independent (37 CFR 1.16(h))	* 3	Minus	*** 3	= 0	x =		OR	x =	0
Application Size Fee (37 CFR 1.16(s))									
FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM (37 CFR 1.16(j))					N/A		OR	N/A	
					TOTAL ADD'L FEE		OR	TOTAL ADD'L FEE	0

AMENDMENT B

	(Column 1) CLAIMS REMAINING AFTER AMENDMENT		(Column 2) HIGHEST NUMBER PREVIOUSLY PAID FOR	(Column 3) PRESENT EXTRA	SMALL ENTITY RATE ($)	SMALL ENTITY ADDITIONAL FEE ($)	OR	OTHER THAN SMALL ENTITY RATE ($)	OTHER THAN SMALL ENTITY ADDITIONAL FEE ($)
Total (37 CFR 1.16(i))	*	Minus	**	=	x =		OR	x =	
Independent (37 CFR 1.16(h))	*	Minus	***	=	x =		OR	x =	
Application Size Fee (37 CFR 1.16(s))									
FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM (37 CFR 1.16(j))					N/A		OR	N/A	
					TOTAL ADD'L FEE		OR	TOTAL ADD'L FEE	

* If the entry in column 1 is less than the entry in column 2, write "0" in column 3.
** If the "Highest Number Previously Paid For" IN THIS SPACE is less than 20, enter "20".
*** If the "Highest Number Previously Paid For" IN THIS SPACE is less than 3, enter "3".
The "Highest Number Previously Paid For" (Total or Independent) is the highest number found in the appropriate box in column 1.